5

UNITED STATES PATENT AND TRADEMARK OFFICE

UNITED STATES DEPARTMENT OF COMMERCE
United States Patent and Trademark Office
Address: COMMISSIONER FOR PATENTS
P.O. Box 1450
Alexandria, Virginia 22313-1450
www.uspto.gov

APPLICATION NO.	FILING DATE	FIRST NAMED INVENTOR	ATTORNEY DOCKET NO.	CONFIRMATION NO.
10/772,992	02/05/2004	James S. Miller	13768.493	5389

47973 7590 07/12/2007
WORKMAN NYDEGGER/MICROSOFT
1000 EAGLE GATE TOWER
60 EAST SOUTH TEMPLE
SALT LAKE CITY, UT 84111

EXAMINER	
WANG, BEN C	
ART UNIT	PAPER NUMBER
2192	
MAIL DATE	DELIVERY MODE
07/12/2007	PAPER

Please find below and/or attached an Office communication concerning this application or proceeding.

The time period for reply, if any, is set in the attached communication.

Office Action Summary	**Application No.** 10/772,992	**Applicant(s)** MILLER ET AL.	
	Examiner Ben C. Wang	**Art Unit** 2192	

-- The MAILING DATE of this communication appears on the cover sheet with the correspondence address --

Period for Reply

A SHORTENED STATUTORY PERIOD FOR REPLY IS SET TO EXPIRE 3 MONTH(S) OR THIRTY (30) DAYS, WHICHEVER IS LONGER, FROM THE MAILING DATE OF THIS COMMUNICATION.

- Extensions of time may be available under the provisions of 37 CFR 1.136(a). In no event, however, may a reply be timely filed after SIX (6) MONTHS from the mailing date of this communication.
- If NO period for reply is specified above, the maximum statutory period will apply and will expire SIX (6) MONTHS from the mailing date of this communication.
- Failure to reply within the set or extended period for reply will, by statute, cause the application to become ABANDONED (35 U.S.C. § 133). Any reply received by the Office later than three months after the mailing date of this communication, even if timely filed, may reduce any earned patent term adjustment. See 37 CFR 1.704(b).

Status

1)☒ Responsive to communication(s) filed on *24 April 2007*.

2a)☒ This action is **FINAL**. 2b)☐ This action is non-final.

3)☐ Since this application is in condition for allowance except for formal matters, prosecution as to the merits is closed in accordance with the practice under *Ex parte Quayle*, 1935 C.D. 11, 453 O.G. 213.

Disposition of Claims

4)☒ Claim(s) *1-27* is/are pending in the application.

4a) Of the above claim(s) _____ is/are withdrawn from consideration.

5)☐ Claim(s) _____ is/are allowed.

6)☒ Claim(s) *1-27* is/are rejected.

7)☐ Claim(s) _____ is/are objected to.

8)☐ Claim(s) _____ are subject to restriction and/or election requirement.

Application Papers

9)☐ The specification is objected to by the Examiner.

10)☐ The drawing(s) filed on _____ is/are: a)☐ accepted or b)☐ objected to by the Examiner.

Applicant may not request that any objection to the drawing(s) be held in abeyance. See 37 CFR 1.85(a).

Replacement drawing sheet(s) including the correction is required if the drawing(s) is objected to. See 37 CFR 1.121(d).

11)☐ The oath or declaration is objected to by the Examiner. Note the attached Office Action or form PTO-152.

Priority under 35 U.S.C. § 119

12)☐ Acknowledgment is made of a claim for foreign priority under 35 U.S.C. § 119(a)-(d) or (f).

a)☐ All b)☐ Some * c)☐ None of:

1.☐ Certified copies of the priority documents have been received.

2.☐ Certified copies of the priority documents have been received in Application No. _____.

3.☐ Copies of the certified copies of the priority documents have been received in this National Stage application from the International Bureau (PCT Rule 17.2(a)).

* See the attached detailed Office action for a list of the certified copies not received.

Attachment(s)

1) ☒ Notice of References Cited (PTO-892)	4) ☐ Interview Summary (PTO-413) Paper No(s)/Mail Date. _____ .
2) ☐ Notice of Draftsperson's Patent Drawing Review (PTO-948)	5) ☐ Notice of Informal Patent Application
3) ☐ Information Disclosure Statement(s) (PTO/SB/08) Paper No(s)/Mail Date _____.	6) ☐ Other: _____.

DETAILED ACTION

1. Applicant's amendment dated April 24, 2007, responding to the January 25, 2006 Office action provided in the rejection of claims 1-27, wherein claims 1, 7, 20-22, and 26-27 have been amended, claims 2-6, 8-19, 23-25 are remained as original.

Claims 1-27 remain pending in the application and which have been fully considered by the examiner.

Applicant's arguments with respect to claims rejection have been fully considered but are moot in view of the new grounds of rejection – see both M. Gunderloy and *S. Pratschner* arts made of record, as applied hereto.

Applicant's amendment necessitated the new ground(s) of rejection presented in this Office action. Accordingly, **THIS ACTION IS MADE FINAL**. See MPEP § 706.07(a). Applicant is reminded of the extension of time policy as set forth in 37 CFR 1.136(a).

A shortened statutory period for reply to this final action is set to expire THREE MONTHS from the mailing date of this action. In the event a first reply is filed within TWO MONTHS of the mailing date of this final action and the advisory action is not mailed until after the end of the THREE-MONTH shortened statutory period, then the shortened statutory period will expire on the date the advisory action is mailed, and any extension fee pursuant to 37 CFR 1.136(a) will be calculated from the mailing date of the advisory action. In no event, however, will the statutory period for reply expire later than SIX MONTHS from the date of this final action.

Claim Rejections – 35 USC § 102(b)

The following is a quotation of the appropriate paragraphs of 35 U.S.C. 102(b) that form the basis for the rejections under this section made in this office action:

> A person shall be entitled to a patent unless –
>
> (b) the invention was patented or described in a printed publication in this or a foreign country or in public use or on sale in this country, more than one year prior to the date of application for patent in the United States.

2. Claims 1-6, 9, 12-15, 20-22, 24, and 26-27 are rejected under 35 U.S.C. 102(b) as being anticipated by M. Gunderloy (*Managing Versions of an Application, Feb. 2002, Lark Group, Inc., pp. 1-6*) (hereinafter 'Gunderloy' - art made of record)

3. **As to claim 1** (Currently Amended), Gunderloy discloses in a computerized system that includes one or more program computer-executable components including one or more computer-executable requesting components configured to execute one or more computer-executable target components in the computerized system, a method of providing a computer-executable requesting component with access to an appropriate version of a computer-executable target component upon request, comprising the acts of: receiving a request from a requesting component for access by the requesting component of a specified version of a computer-executable target component; upon receiving the request from the requesting component (e.g., P. 3, Sec. of "Practice Modifying Version Information at Runtime", 1st Par. – you'll create two versions of a class library and a client application (requesting component) that call a method from one version of the library); identifying a versioning policy of the specified version of the requested target component; identifying an appropriate version of the target component

based on the versioning policy of the specified target component; and providing the requesting component with access to the appropriate version of the target component, wherein the requesting component executes the identified and provided target component (e.g., P. 3, Sec. of "Practice Modifying Version Information at Runtime", 1st Par. – you'll see that even though there is a more recent version of the library installed, the older version will be used by the client application until you explicitly construct an application configuration file; P. 3, Sec. of "Finding the Correct Version at Runtime" – when you run a Microsoft .NET™ application that uses an external component, the Microsoft .NET™ runtime checks four places to determine which version of the component to load: 1) the original version from the assembly manifest is the default version to load, 2) the runtime then checks for the presence of an application policy file that overrides the version information for this application only, 3) the runtime then checks for the presence of a publisher policy file that overrides the version information for this component in all applications, 4) the runtime then checks for the presence of an administrator policy file that overrides the version information for this component system-wide; P. 3, Sec. of "Policy Files" – there are three policy files that the runtime checks for version information 1) the application policy files has the same name as the application plus the extension ".config" and resides in the same directory as the application, 2) the publisher policy file is distributed by a component publisher together with a new version of a component, 3) the machine configuration file is named *Machine.config* and is stored in the Config. directory beneath the directory where the Microsoft .NET™ runtime is installed; P. 5, Sec. of "Summary", - Microsoft .NET™

allows very fine control of version information. By default, an assembly will use only the versions of components that were available when the assembly was compiled. But as the developer of an application, the publisher of a component, or the administrator of a system you can override this policy and specify the versions to be used on a component-by-component basis).

4. **As to claim 20** (Currently Amended), Gunderloy discloses in a computerized system that includes one or more program computer-executable components including one or more computer-executable requesting components that can request to access one or more computer-executable target components in the computerized system, a method of providing a computer-executable requesting component with access to an appropriate version of a target component, comprising: receiving a request from a requesting component for access by the requesting component of a specified version of a target component (e.g., P. 3, Sec. of "Practice Modifying Version Information at Runtime", 1st Par. – you'll create two versions of a class library and a client application (requesting component) that call a method from one version of the library); a step for, upon receiving the request form the requesting component, determining an appropriate version of the requested target component based on a versioning policy corresponding to the requested target component (e.g., P. 3, Sec. of "Practice Modifying Version Information at Runtime", 1st Par. – you'll see that even though there is a more recent version of the library installed, the older version will be used by the client application until you explicitly construct an application configuration file; P. 3, Sec. of "Finding the

Correct Version at Runtime" – when you run a Microsoft .NET™ application that uses an external component, the Microsoft .NET™ runtime checks four places to determine which version of the component to load: 1) the original version from the assembly manifest is the default version to load, 2) the runtime then checks for the presence of an application policy file that overrides the version information for this application only, 3) the runtime then checks for the presence of a publisher policy file that overrides the version information for this component in all applications, 4) the runtime then checks for the presence of an administrator policy file that overrides the version information for this component system-wide; P. 3, Sec. of "Policy Files" – there are three policy files that the runtime checks for version information 1) the application policy files has the same name as the application plus the extension ".config" and resides in the same directory as the application, 2) the publisher policy file is distributed by a component publisher together with a new version of a component, 3) the machine configuration file is named *Machine.config* and is stored in the Config. directory beneath the directory where the Microsoft .NET™ runtime is installed; P. 5, Sec. of "Summary", - Microsoft .NET™ allows very fine control of version information. By default, an assembly will use only the versions of components that were available when the assembly was compiled. But as the developer of an application, the publisher of a component, or the administrator of a system you can override this policy and specify the versions to be used on a component-by-component basis), and allowing access to an appropriate version of the requested target component such that the requesting component accesses the appropriate target component as it has been configured to do so, and such that the requesting component does not fail when requesting access to a component that has been upgraded (e.g., P. 2, Sec. of "Version Compatibility", 1st Par. – the main use of version information is to help applications and developers determine whether a particular version of a component is compatible with an application that makes use of the component; by default, Microsoft .NET™ uses only the versions of components that were present when an assembly was compiled; 2nd Par. – for example, if a Visual Basic™ 6.0 application calls a component named *xyz.dll*, and version 1 was present on the computer at the time that the application was compiled, and later version 1.1 of *xyz.dll* is installed, the application will automatically call version 1.1).

5. **As to claim 22** (Currently Amended), Gunderloy discloses in a computerized system that includes one or more program components including one or more requesting components that can request to access one or more target components in the computerized system, a method of upgrading a computer executable target component such that a computer-executable requesting component that accesses the target component continues to operate effectively after the target component has been upgraded, comprising the acts of: identifying that a requesting component is configured to execute a computer-executable target component (e.g., P. 3, Sec. of "Practice Modifying Version Information at Runtime", 1st Par. – you'll create two versions of a class library and a client application (requesting component) that call a method from one version of the library); identifying a versioning policy in at least an existing version of the target component and a versioning policy in a previously installed version of the target

component; and identifying which versions of the target component should remain on the system based on any of the identified versioning policies corresponding to at least the existing version of the target component and the previously installed version of the target component (e.g., P. 3, Sec. of "Practice Modifying Version Information at Runtime", 1st Par. – you'll see that even though there is a more recent version of the library installed, the older version will be used by the client application until you explicitly construct an application configuration file; P. 3, Sec. of "Finding the Correct Version at Runtime" – when you run a Microsoft .NET™ application that uses an external component, the Microsoft .NET™ runtime checks four places to determine which version of the component to load: 1) the original version from the assembly manifest is the default version to load, 2) the runtime then checks for the presence of an application policy file that overrides the version information for this application only, 3) the runtime then checks for the presence of a publisher policy file that overrides the version information for this component in all applications, 4) the runtime then checks for the presence of an administrator policy file that overrides the version information for this component system-wide; P. 3, Sec. of "Policy Files" – there are three policy files that the runtime checks for version information 1) the application policy files has the same name as the application plus the extension ".config" and resides in the same directory as the application, 2) the publisher policy file is distributed by a component publisher together with a new version of a component, 3) the machine configuration file is named *Machine.config* and is stored in the Config. directory beneath the directory where the Microsoft .NET™ runtime is installed; P. 5, Sec. of "Summary", - Microsoft .NET™

allows very fine control of version information. By default, an assembly will use only the versions of components that were available when the assembly was compiled. But as the developer of an application, the publisher of a component, or the administrator of a system you can override this policy and specify the versions to be used on a component-by-component basis).

6. **As to claim 26** (Currently Amended), Gunderloy discloses in a computerized system including one or more requesting components that are configured to access one or more target components in the computerized system, a computer storage product having computer-executable instructions stored thereon that, when executed, cause one or more processors in the computerized system to execute a method of providing a computer-executable requesting component upon request, comprising the acts of: receiving a request from a requesting component for access by the requesting component of a specified version of a computer-executable target component; upon receiving the request from the requesting component (e.g., P. 3, Sec. of "Practice Modifying Version Information at Runtime", 1st Par. – you'll create two versions of a class library and a client application (requesting component) that call a method from one version of the library), identifying a versioning policy of the specified version of the requested target component; identifying an appropriate version of the target component based on the versioning policy of the specified target component; and providing the requesting component with access to the appropriate version of the target component, wherein the requesting component executes the identified and provided target

component (e.g., P. 3, Sec. of "Practice Modifying Version Information at Runtime", 1st Par. – you'll see that even though there is a more recent version of the library installed, the older version will be used by the client application until you explicitly construct an application configuration file; P. 3, Sec. of "Finding the Correct Version at Runtime" – when you run a Microsoft .NET™ application that uses an external component, the Microsoft .NET™ runtime checks four places to determine which version of the component to load: 1) the original version from the assembly manifest is the default version to load, 2) the runtime then checks for the presence of an application policy file that overrides the version information for this application only, 3) the runtime then checks for the presence of a publisher policy file that overrides the version information for this component in all applications, 4) the runtime then checks for the presence of an administrator policy file that overrides the version information for this component system-wide; P. 3, Sec. of "Policy Files" – there are three policy files that the runtime checks for version information 1) the application policy files has the same name as the application plus the extension ".config" and resides in the same directory as the application, 2) the publisher policy file is distributed by a component publisher together with a new version of a component, 3) the machine configuration file is named *Machine.config* and is stored in the Config. directory beneath the directory where the Microsoft .NET™ runtime is installed; P. 5, Sec. of "Summary", - Microsoft .NET™ allows very fine control of version information. By default, an assembly will use only the versions of components that were available when the assembly was compiled. But as the developer of an application, the publisher of a component, or the administrator of a system you can override this policy and specify the versions to be used on a component-by-component basis).

7. **As to claim 27** (Currently Amended), Gunderloy discloses in a computerized system including one or more requesting components that are configured to access one or more target components in the computerized system, a computer program storage product having computer-executable instructions stored thereon that, when executed, cause one or more processors in the computerized system to execute a method of upgrading a computer-executable target component such that a computer-executable requesting component that accesses the computer-executable target component continues to operate effectively after the target component has been upgraded, comprising the acts of: identifying that a requesting component is configured to execute a computer-executable target component (e.g., P. 3, Sec. of "Practice Modifying Version Information at Runtime", 1st Par. – you'll create two versions of a class library and a client application (requesting component) that call a method from one version of the library); identifying a versioning policy in at least an existing version of the target component and a versioning policy in a previously installed version of the target component; and identifying which versions of the target component should remain on the system based on any of the identified versioning policies corresponding to at least the existing version of the target component and the previously installed version of the target component (e.g., P. 3, Sec. of "Practice Modifying Version Information at Runtime", 1st Par. – you'll see that even though there is a more recent version of the

library installed, the older version will be used by the client application until you explicitly construct an application configuration file; P. 3, Sec. of "Finding the Correct Version at Runtime" – when you run a Microsoft .NET™ application that uses an external component, the Microsoft .NET™ runtime checks four places to determine which version of the component to load: 1) the original version from the assembly manifest is the default version to load, 2) the runtime then checks for the presence of an application policy file that overrides the version information for this application only, 3) the runtime then checks for the presence of a publisher policy file that overrides the version information for this component in all applications, 4) the runtime then checks for the presence of an administrator policy file that overrides the version information for this component system-wide; P. 3, Sec. of "Policy Files" – there are three policy files that the runtime checks for version information 1) the application policy files has the same name as the application plus the extension ".config" and resides in the same directory as the application, 2) the publisher policy file is distributed by a component publisher together with a new version of a component, 3) the machine configuration file is named *Machine.config* and is stored in the *Config.* directory beneath the directory where the Microsoft .NET™ runtime is installed; P. 5, Sec. of "Summary", - Microsoft .NET™ allows very fine control of version information. By default, an assembly will use only the versions of components that were available when the assembly was compiled. But as the developer of an application, the publisher of a component, or the administrator of a system you can override this policy and specify the versions to be used on a component-by-component basis).

8. **As to claim 2** (Original) (incorporating the rejection in claim 1), Gunderloy discloses the method wherein the requested version of the target component is one of a library component and a platform component (e.g., P. 3, Sec. of "Practice Modifying Version Information at Runtime", 1st Par. through P. 5, Sec. of "Try It Out"; P. 3, Sec. of "Policy Files").

9. **As to claim 3** (Original) (incorporating the rejection in claim 1), Gunderloy discloses the method wherein identifying an appropriate version of the target component comprises identifying a more recent version of the target component in response to a request for an earlier version of the target even though the more recent version and the earlier version are both accessible to the computerized system (e.g., P. 2, Sec. of "Version Compatibility", 1st Par. – the main use of version information is to help applications and developers determine whether a particular version of a component is compatible with an application that makes use of the component; by default, Microsoft .NET™ uses only the versions of components that were present when an assembly was compiled; 2nd Par. – for example, if a Visual Basic™ 6.0 application calls a component named *xyz.dll*, and <u>version 1 was present on the computer at the time that the application was compiled, and later version 1.1 of *xyz.dll* is installed, the application will automatically call version 1.1</u>).

10. **As to claim 4** (Original) (incorporating the rejection in claim 2), Gunderloy discloses the method identifying a more recent version of the target component in response to a request for an earlier version of the target even though the more recent version and the earlier version are both accessible to the computerized system comprises identifying a more recent version of a platform component even though an earlier version of the platform component remained on the system when the more recent version was received at the computerized system (e.g., P. 2, Sec. of "Version Compatibility", 1st Par. – the main use of version information is to help applications and developers determine whether a particular version of a component is compatible with an application that makes use of the component; by default, Microsoft .NET™ uses only the versions of components that were present when an assembly was compiled; 2nd Par. – for example, if a Visual Basic™ 6.0 application calls a component named *xyz.dll*, and version 1 was present on the computer at the time that the application was compiled, and later version 1.1 of *xyz.dll* is installed, the application will automatically call version 1.1).

11. **As to claim 5** (Original) (incorporating the rejection in claim 1), Gunderloy discloses the method wherein the versioning policy of the specified version of the target component is identified when the target component is one or more of compiled, configured, installed, and run on the computerized system (e.g., P. 3, Sec. of "Practice Modifying Version Information at Runtime", 1st Par. – you'll see that even though there is a more recent version of the library installed, the older version will be used by the client

application until you explicitly construct an application configuration file; P. 3, Sec. of "Finding the Correct Version at Runtime" – when you run a Microsoft .NET™ application that uses an external component, the Microsoft .NET™ runtime checks four places to determine which version of the component to load: 1) the original version from the assembly manifest is the default version to load, 2) the runtime then checks for the presence of an application policy file that overrides the version information for this application only, 3) the runtime then checks for the presence of a publisher policy file that overrides the version information for this component in all applications, 4) the runtime then checks for the presence of an administrator policy file that overrides the version information for this component system-wide; P. 3, Sec. of "Policy Files" – there are three policy files that the runtime checks for version information 1) the application policy files has the same name as the application plus the extension ".config" and resides in the same directory as the application, 2) the publisher policy file is distributed by a component publisher together with a new version of a component, 3) the machine configuration file is named *Machine.config* and is stored in the Config. directory beneath the directory where the Microsoft .NET™ runtime is installed; P. 5, Sec. of "Summary", - Microsoft .NET™ allows very fine control of version information. By default, an assembly will use only the versions of components that were available when the assembly was compiled. But as the developer of an application, the publisher of a component, or the administrator of a system you can override this policy and specify the versions to be used on a component-by-component basis).

12. **As to claim 6** (Original) (incorporating the rejection in claim 1), Gunderloy discloses the method wherein version information that identifies the specified target component is stored in the requesting component when the requesting component is one or more of compiled, configured, installed, and run on the computerized system (P. 1, Sec. of "Setting Version Numbers", 1st Par. – when you create a new application using Visual Basic .NET™, a module named *AssemblyInfo.vb* is automatically added to you project. This module contains setting for assembly attributes, the Microsoft .NET™ equivalent of project properties).

13. **As to claim 9** (Original) (incorporating the rejection in claim 1), Gunderloy discloses the method wherein the appropriate version of the target component is different from the version of the target component that was requested (e.g., P. 2, Sec. of "Version Compatibility", 1st Par. – the main use of version information is to help applications and developers determine whether a particular version of a component is compatible with an application that makes use of the component; by default, Microsoft .NET™ uses only the versions of components that were present when an assembly was compiled; 2nd Par. – for example, if a Visual Basic™ 6.0 application calls a component named *xyz.dll*, and version 1 was present on the computer at the time that the application was compiled, and later version 1.1 of *xyz.dll* is installed, the application will automatically call version 1.1).

14. **As to claim 12** (Original) (incorporating the rejection in claim 1), Gunderloy discloses the method wherein the versioning policy is inserted into computer-executable instructions in the target component prior to one of installing, configuring, and executing the target component on the computerized system (e.g., P. 3, Sec. of "Policy Files" – policy files are XML files that specify the version of a component to use; the general structure of a policy file follows this template - <configuration> .. </configuration>; P. 2, Sec. of "Version Compatibility", 1st Par. – by default, Microsoft.NET uses only the versions of components that were present when an assembly was compiled; 3rd Par. – by tying assemblies to the versions of components that were present at compile time, the .NET Framework™ helps to avoid the "DLL Hell" syndrome in which new versions of shared libraries break existing application).

15. **As to claim 13** (Original) (incorporating the rejection in claim 1), Gunderloy discloses the method wherein the versioning policy is further identified in any version of the target component (e.g., P. 3, Sec. of "Policy Files" – the version for the *oldVersion* attribute can either be a single version number such as "1.0.0.0").

16. **As to claim 14** (Original) (incorporating the rejection in claim 12), Gunderloy discloses the method wherein the versioning policy identifies that any of the prior version of the target component and the more recent version of the target component is configured to be accessed by a specific version of the requesting component (e.g., P. 3, Sec. of "Policy Files" – the version for the *oldVersion* attribute can either be a single

version number such as "1.0.0.0" or a range of version numbers such as "1.0.0.0-3.0.0.0").

17. **As to claim 15** (Original) (incorporating the rejection in claim 1), Gunderloy discloses the method further comprising identifying a component scope that is associated with the target component (e.g., P. 3, Sec. of "Policy Files" – the version for the *oldVersion* attribute can either be a single version number such as "1.0.0.0" or a range of version numbers such as "1.0.0.0-3.0.0.0").

18. **As to claim 21** (Currently Amended) (incorporating the rejection in claim 20), Gunderloy discloses the method wherein the step for allowing access to an appropriate version of the requested target component comprises the corresponding acts of: upon receiving the request from the requesting component, identifying a versioning policy of the specified version of the requested target component; identifying an appropriate version of the target component based on the versioning policy of the specified target component; providing the requesting component with access to the appropriate version of the target component, wherein the requesting component executes the identified and provided target component (e.g., P. 3, Sec. of "Practice Modifying Version Information at Runtime", 1st Par. – you'll see that even though there is a more recent version of the library installed, the older version will be used by the client application until you explicitly construct an application configuration file; P. 3, Sec. of "Finding the Correct Version at Runtime" – when you run a Microsoft .NET™ application that uses an external

component, the Microsoft .NET™ runtime checks four places to determine which version of the component to load: 1) the original version from the assembly manifest is the default version to load, 2) the runtime then checks for the presence of an application policy file that overrides the version information for this application only, 3) the runtime then checks for the presence of a publisher policy file that overrides the version information for this component in all applications, 4) the runtime then checks for the presence of an administrator policy file that overrides the version information for this component system-wide; P. 3, Sec. of "Policy Files" – there are three policy files that the runtime checks for version information 1) the application policy files has the same name as the application plus the extension ".config" and resides in the same directory as the application, 2) the publisher policy file is distributed by a component publisher together with a new version of a component, 3) the machine configuration file is named *Machine.config* and is stored in the Config. directory beneath the directory where the Microsoft .NET™ runtime is installed; P. 5, Sec. of "Summary", - Microsoft .NET™ allows very fine control of version information. By default, an assembly will use only the versions of components that were available when the assembly was compiled. But as the developer of an application, the publisher of a component, or the administrator of a system you can override this policy and specify the versions to be used on a component-by-component basis).

19. **As to claim 24** (Original) (incorporating the rejection in claim 22), Gunderloy discloses the method wherein if the versioning policy indicates that the requesting

component is a library component, adding the existing version of the target component to the system without removing the previously installed version of the target component (e.g., P. 3, Sec. of "Practice Modifying Version Information at Runtime", 1st par. – you'll create two versions of a class library and a client application that calls a method from one version of the library).

Claim Rejections – 35 USC § 103(a)

The following is a quotation of 35 U.S.C. 103(a) which forms the basis for all obviousness rejections set forth in this office action:

> (a) A patent may not be obtained though the invention is not identically disclosed or described as set forth in section 102 of this title, if the differences between the subject matter sought to be patented and the prior art are such that the subject matter as a whole would have been obvious at the time the invention was made to a person having ordinary skill in the art to which said subject matter pertains. Patentability shall not be negatived by the manner in which the invention was made.

20. Claims 7-8, 10-11, 16-19, 23, and 25 are rejected under 35 U.S.C. 103(a) as being unpatentable over Gunderloy in view of S. Pratschner (*Simplifying Deployment and Solving DLL Hell with the .NET Framework™, Nov. 2001, Microsoft Corporation™, pp. 1-12*) (hereinafter 'Pratschner' - art made of record)

21. **As to claim 7** (Currently Amended) (incorporating the rejection in claim 1), Gunderloy does not explicitly disclose the method further comprising: identifying one or more requesting components that are able to access a prior version of the target component; identifying that none of the one or more requesting components are

configured to request the prior version of the target component; and deleting the prior version of the target component.

However, in an analogous art of Simplifying Deployment and Solving DLL Hell with the .NET Framework™, Pratschner discloses are able to access a prior version of the target component; identifying that none of the one or more requesting components are configured to request the prior version of the target component; and deleting the prior version of the target component (e.g., P. 1, Sec. of "Introduction", 3rd Par. - .NET™ uses assemblies to solve versioning and deployment problems; in particular, how assemblies are structured, how they are named, and how compilers and the Common Language Runtime™ (CLR) use assemblies to record and enforce version dependencies between pieces of applications, and how applications and administrators can customize versioning behavior through what we call *version policies;* P. 7, Sec. of "Version Policy"' P. 8, Sec. of "Default Version Policy", Sec. of "Custom Version Policy", and Sec. of "Version Policy Levels" – including "Application –specific Policy", "Publisher Policy", and "Machine-wide Policy"; P. 9, Sec. of "Bypassing Publisher Policy").

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Pratschner into the Gunderloy's system to further provide the method identifying one or more requesting components that are able to access a prior version of the target component; identifying that none of the one or more requesting components are configured to request the prior version of the target component; and deleting the prior version of the target component in Gunderloy system.

The motivation is that it would further enhance the Gunderloy's system by taking, advancing and/or incorporating Pratschner's system which offers significant advantages that the CLR records version information between pieces of an application and uses that information at run time to ensure that the proper version of a dependency is loaded; version policies can be used by both application developers and administrators to provide some flexibility in choosing which version of a given assembly is loaded as once suggested by Pratschner (e.g., P. 12, Sec. of "Summary").

22. **As to claim 8** (Original) (incorporating the rejection in claim 1), Gunderloy does not explicitly disclose the method wherein the appropriate version of the target component is the version of the target component that was requested.

However, in an analogous art of Simplifying Deployment and Solving DLL Hell with the .NET Framework™, Pratschner discloses the method wherein the appropriate version of the target component is the version of the target component that was requested (e.g., P. 8, Sec. of "Default Version Policy" – when resolving a reference, the CLR takes the version from the calling assembly's manifest and loads the version of the dependency <u>with the exact same version number;</u> in this way, the caller gets <u>the exact version that he was built and tested against</u>).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Pratschner into the Gunderloy's system to further provide the method wherein the appropriate version of the target component is the version of the target component that was requested in Gunderloy system.

The motivation is that it would further enhance the Gunderloy's system by taking, advancing and/or incorporating Pratschner's system which offers significant advantages that the CLR records version information between pieces of an application and uses that information at run time to ensure that the proper version of a dependency is loaded; version policies can be used by both application developers and administrators to provide some flexibility in choosing which version of a given assembly is loaded as once suggested by Pratschner (e.g., P. 12, Sec. of "Summary").

23. **As to claim 10** (Original) (incorporating the rejection in claim 8), Gunderloy does not explicitly disclose the method wherein target component access is provided to the requesting component through a determining module.

However, in an analogous art of Simplifying Deployment and Solving DLL Hell with the .NET Framework™, Pratschner discloses the method wherein target component access is provided to the requesting component through a determining module (e.g., P. 1, Sec. of "Introduction", 3rd Par. – the Common Language Runtime™ (CLR) uses assemblies to record and enforce version dependencies between pieces of an application; P. 5, Sec. of "Application-Private Assemblies, 2nd Par. – 3rd Par. – the CLR finds these assemblies through a process called probing; probing is simply a mapping of the assembly name to the name of the file that contains the manifest; specifically, the CLR™ takes the name of the assembly recorded in the assembly reference, appends ".dll" and looks for that file in the application directory; P. 8, Sec. of "Default Version Policy" – when resolving a reference, the CLR™ takes the version from the calling assembly's manifest and loads the version of the dependency with the exact same version number; the first thing the CLR™ does when binding to a strongly named assembly is determine which version of the assembly to bind to).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Pratschner into the Gunderloy's system to further provide the method wherein target component access is provided to the requesting component through a determining module in Gunderloy system.

The motivation is that it would further enhance the Gunderloy's system by taking, advancing and/or incorporating Pratschner's system which offers significant advantages that the CLR™ records version information between pieces of an application and uses that information at run time to ensure that the proper version of a dependency is loaded; version policies can be used by both application developers and administrators to provide some flexibility in choosing which version of a given assembly is loaded as once suggested by Pratschner (e.g., P. 12, Sec. of "Summary").

24. **As to claim 11** (Original) (incorporating the rejection in claim 9), Pratschner discloses the method wherein the availability of one or more of the prior version of the target component and the more recent version of the target component is identified by a determining module when the one or more of the prior version of the target component and the more recent version of the target component is received by the computerized system (e.g., P. 1, Sec. of "Introduction", 3rd Par. – the Common Language Runtime™

(CLR) uses assemblies to record and enforce version dependencies between pieces of an application; P. 5, Sec. of "Application-Private Assemblies, 2nd Par. – 3rd Par. – the CLR™ finds these assemblies through a process called probing; probing is simply a mapping of the assembly name to the name of the file that contains the manifest; specifically, the CLR™ takes the name of the assembly recorded in the assembly reference, appends ".dll" and looks for that file in the application directory; P. 8, Sec. of "Default Version Policy" – when resolving a reference, the CLR™ takes the version from the calling assembly's manifest and loads the version of the dependency with the exact same version number; the first thing the CLR™ does when binding to a strongly named assembly is determine which version of the assembly to bind to).

25. **As to claim 16** (Original) (incorporating the rejection in claim 1), Gunderloy does not explicitly disclose the method wherein access to the specified version of the target component is further based on one of the identified component scope associated with the target component, and a target component scope supplied by a system administrator.

However, in an analogous art of Simplifying Deployment and Solving DLL Hell with the .NET Framework™, Pratschner discloses the method wherein access to the specified version of the target component is further based on one of the identified component scope associated with the target component, and a target component scope supplied by a system administrator (P. 10, Fig. 6 – The Admin Tool; P. 9, Sec. of "Setting Version Policies with the .NET Configuration Tool – the .NET Framework ships with a graphical admin tool so you don't have to worry about hand editing XML policy files; the tool support both application and machine-wide version policy; you'll find the tool in the Administrative Tools section of Control Panel; the initial screen of the admin tool looks like Fig. 6).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Pratschner into the Gunderloy's system to further provide the method wherein access to the specified version of the target component is further based on one of the identified component scope associated with the target component, and a target component scope supplied by a system administrator in Gunderloy system.

The motivation is that it would further enhance the Gunderloy's system by taking, advancing and/or incorporating Pratschner's system which offers significant advantages that the CLR records version information between pieces of an application and uses that information at run time to ensure that the proper version of a dependency is loaded; version policies can be used by both application developers and administrators to provide some flexibility in choosing which version of a given assembly is loaded as once suggested by Pratschner (e.g., P. 12, Sec. of "Summary").

26. **As to claim 17** (Original) (incorporating the rejection in claim 15), Pratschner discloses the method wherein the identified component scope specifies that access to the specified version of the target component is provided in one or more of a machine level, a process level, and a sub-process level (e.g., P. 4, Sec. of "Versioning and

Sharing", 4th Par. – in .NET, sharing code between applications is an explicit decision; assemblies that are shared have some additional requirements; specifically, shared assemblies should support side by side so multiple versions of the same assembly can be installed and run on the same machine, or even within the same process, at the same time; P. 8, Sec. of "Version Policy Levels" – including "Application-specific Policy", Publisher Policy", and "Machine-wide Policy"; P. 9, Sec. of "Machine-wide Policy" – the final policy level is machine-wide policy; machine-wide policy is stored in *machine.config* which is located in the "config" subdirectory under the .NET Framework install directory).

27. **As to claim 18** (Original) (incorporating the rejection in claim 1), Gunderloy does not explicitly disclose the method wherein the requested target component is a library component, the method further comprising identifying a servicing value associated with the requested target component.

However, in an analogous art of Simplifying Deployment and Solving DLL Hell with the .NET Framework™, Pratschner discloses the method wherein the requested target component is a library component, the method further comprising identifying a servicing value associated with the requested target component (e.g., P. 8, Sec. of "Custom Version Policy", the vendor of a shared assembly may have shipped a service release to an existing assembly and wou8ld like all applications to being using the service release instead of the original version; these scenarios and others are supported in the .NET Framework™ through version policies; P. 11, Sec. of

"Deployment" – the .NET Framework™ provides extensive code download services are integration with Windows Installer).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Pratschner into the Gunderloy's system to further provide the method wherein the requested target component is a library component, the method further comprising identifying a servicing value associated with the requested target component in Gunderloy system.

The motivation is that it would further enhance the Gunderloy's system by taking, advancing and/or incorporating Pratschner's system which offers significant advantages that the CLR records version information between pieces of an application and uses that information at run time to ensure that the proper version of a dependency is loaded; version policies can be used by both application developers and administrators to provide some flexibility in choosing which version of a given assembly is loaded as once suggested by Pratschner (e.g., P. 12, Sec. of "Summary").

28. **As to claim 19** (Original) (incorporating the rejection in claim 1), Pratschner discloses the method wherein identifying an appropriate version of the target component comprising identifying an updated version of a library component based on the identified versioning policy and the identified servicing value (e.g., P. 7, Sec. of "Version Policy" – including "Default Version Policy", "Custom Version Policy", and Version Policy Levels"; P. 8, Sec. of "Custom Version Policy", the vendor of a shared assembly may have shipped a service release to an existing assembly and wou8ld like

all applications to being using the service release instead of the original version; these scenarios and others are supported in the .NET Framework™ through version policies; P. 11, Sec. of "Deployment" – the .NET Framework™ provides extensive code download services are integration with Windows Installer).

29. **As to claim 23** (Original) (incorporating the rejection in claim 22), Gunderloy does not explicitly disclose the method further comprising receiving an updated version of the target component over a network from a network service provider.

However, in an analogous art of Simplifying Deployment and Solving DLL Hell with the .NET Framework™, Pratschner discloses the method further comprising receiving an updated version of the target component over a network from a network service provider (e.g., P. 11, Sec. of "Deployment" – deployment involves at least two different aspects: packaging the code, and distributing the packages to the various clients and servers on which the application will run; a primary goal of the .NET Framework™ is to simplify deployment (especially the distribution aspect) by making zero-impact install and *xcopy* deployment feasible).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Pratschner into the Gunderloy's system to further provide the method further comprising receiving an updated version of the target component over a network from a network service provider in Gunderloy system.

The motivation is that it would further enhance the Gunderloy's system by taking, advancing and/or incorporating Pratschner's system which offers significant advantages that the CLR™ records version information between pieces of an application and uses that information at run time to ensure that the proper version of a dependency is loaded; version policies can be used by both application developers and administrators to provide some flexibility in choosing which version of a given assembly is loaded as once suggested by Pratschner (e.g., P. 12, Sec. of "Summary").

30. **As to claim 25** (Original) (incorporating the rejection in claim 22), Gunderloy does not explicitly disclose the method wherein if the versioning policy indicates that the requesting component is a platform component, overwriting the previously installed version of the target component with the existing version of the target component.

However, in an analogous art of Simplifying Deployment and Solving DLL Hell with the .NET Framework™, Pratschner discloses the method wherein if the versioning policy indicates that the requesting component is a platform component, overwriting the previously installed version of the target component with the existing version of the target component (e.g., P. 5, Sec. of "Shared Assemblies", 2nd Par. – the .NET Framework™ allows applications and administrators to override the version of an assembly that is used by the application by specifying version policies; P. 9, Sec. of "Policy Evaluation", 2nd par.; P. 9, Sec. of "Bypassing Publisher Policy" – because of the order in which the three types of policy are applied, the publisher's version redirect (publisher policy) can override both the version recorded in the calling assembly's metadata and any application-specific policy that was set).

Therefore, it would have been obvious to one of ordinary skill in the art, at the time the invention was made to combine the teachings of Pratschner into the Gunderloy's system to further provide the method wherein if the versioning policy indicates that the requesting component is a platform component, overwriting the previously installed version of the target component with the existing version of the target component in Gunderloy system.

The motivation is that it would further enhance the Gunderloy's system by taking, advancing and/or incorporating Pratschner's system which offers significant advantages that the CLR™ records version information between pieces of an application and uses that information at run time to ensure that the proper version of a dependency is loaded; version policies can be used by both application developers and administrators to provide some flexibility in choosing which version of a given assembly is loaded as once suggested by Pratschner (e.g., P. 12, Sec. of "Summary").

Conclusion

31. Any inquiry concerning this communication or earlier communications from the examiner should be directed to Ben C. Wang whose telephone number is 571-270-1240. The examiner can normally be reached on Monday - Friday, 8:00 a.m. - 5:00 p.m., EST.

If attempts to reach the examiner by telephone are unsuccessful, the examiner's supervisor, Tuan Q. Dam can be reached on 571-272-3695. The fax phone number for the organization where this application or proceeding is assigned is 571-273-8300.

Information regarding the status of an application may be obtained from the Patent Application Information Retrieval (PAIR) system. Status information for published applications may be obtained from either Private PAIR or Public PAIR. Status information for unpublished applications is available through Private PAIR only. For more information about the PAIR system, see http://pair-direct.uspto.gov. Should you have questions on access to the Private PAIR system, contact the Electronic Business Center (EBC) at 866-217-9197 (toll-free). If you would like assistance from a USPTO Customer Service Representative or access to the automated information system, call 800-786-9199 (IN USA OR CANADA) or 571-272-1000.

[illegible]

TUAN DAM
SUPERVISORY PATENT EXAMINER

BCW [illegible]

June 29, 2007